Parallel Execution
ADMINISTRIVIA

**Project #3** is due Monday October 19th

**Project #4** is due Monday December 10th

**Homework #4** is due Monday November 12th
UPCOMING DATABASE EVENTS

**BlazingDB Tech Talk**
→ Thursday October 25th @ 12pm
→ CIC - 4th floor (ISTC Panther Hollow Room)

**Brytlyt Tech Talk**
→ Thursday November 1st @ 12pm
→ CIC - 4th floor (ISTC Panther Hollow Room)
WHY CARE ABOUT PARALLEL EXECUTION?

Increased performance.
→ Throughput
→ Latency

Increased availability.

Potentially lower TCO.
PARALLEL VS. DISTRIBUTED

Database is spread out across multiple resources to improve parallelism.

Appears as a single database instance to the application.
→ SQL query for a single-node DBMS should generate same result on a parallel or distributed DBMS.
PARALLEL VS. DISTRIBUTED

Parallel DBMSs:
→ Nodes are physically close to each other.
→ Nodes connected with high-speed LAN.
→ Communication cost is assumed to be small.

Distributed DBMSs:
→ Nodes can be far from each other.
→ Nodes connected using public network.
→ Communication cost and problems cannot be ignored.
INTER- VS. INTRA-QUERY PARALLELISM

**Inter-Query**: Different queries are executed concurrently.
→ Increases throughput & reduces latency.

**Intra-Query**: Execute the operations of a single query in parallel.
→ Decreases latency for long-running queries.
TODAY'S AGENDA

Process Models
Execution Parallelism
I/O Parallelism
A DBMS’s **process model** defines how the system is architected to support concurrent requests from a multi-user application.

A **worker** is the DBMS component that is responsible for executing tasks on behalf of the client and returning the results.
PROCESS MODELS

Approach #1: Process per DBMS Worker

Approach #2: Process Pool

Approach #3: Thread per DBMS Worker
PROCESS PER WORKER

Each worker is a separate OS process.
→ Relies on OS scheduler.
→ Use shared-memory for global data structures.
→ A process crash doesn’t take down entire system.
→ Examples: IBM DB2, Postgres, Oracle
A worker uses any process that is free in a pool → Still relies on OS scheduler and shared memory. → Bad for CPU cache locality. → Examples: IBM DB2, Postgres (2015)
Single process with multiple worker threads.
→ DBMS has to manage its own scheduling.
→ May or may not use a dispatcher thread.
→ Thread crash (may) kill the entire system.
→ Examples: IBM DB2, MSSQL, MySQL, Oracle (2014)
Using a multi-threaded architecture has several advantages:
→ Less overhead per context switch.
→ Don’t have to manage shared memory.

The thread per worker model does not mean that you have intra-query parallelism.

I am not aware of any new DBMS built in the last 10 years that doesn’t use threads.
For each query plan, the DBMS has to decide where, when, and how to execute it.
→ How many tasks should it use?
→ How many CPU cores should it use?
→ What CPU core should the tasks execute on?
→ Where should a task store its output?

The DBMS *always* knows more than the OS.
INTER-QUERY PARALLELISM

Improve overall performance by allowing multiple queries to execute simultaneously.

If queries are read-only, then this requires little coordination between queries.

If queries are updating the database at the same time, then this is hard to do this correctly.
→ Need to provide the illusion of isolation.
→ We will discuss more next week.
INTRA-QUERY PARALLELISM

Improve the performance of a single query by executing its operators in parallel.
→ Approach #1: Intra-Operator
→ Approach #2: Inter-Operator

These techniques are not mutually exclusive.

There are parallel algorithms for every relational operator.
INTRA-OPERATOR PARALLELISM

Approach #1: Intra-Operator (Horizontal)
→ Operators are decomposed into independent instances that perform the same function on different subsets of data.

The DBMS inserts an exchange operator into the query plan to coalesce results from children operators.
INTRA-OPERATOR PARALLELISM

```
SELECT A.id, B.value
FROM A, B
WHERE A.id = B.id
AND A.value < 99
AND B.value > 100
```
INTRA-OPERATOR PARALLELISM

```
SELECT A.id, B.value
FROM A, B
WHERE A.id = B.id
AND A.value < 99
AND B.value > 100
```
INTRA-OPERATOR PARALLELISM

SELECT A.id, B.value
FROM A, B
WHERE A.id = B.id
AND A.value < 99
AND B.value > 100
**INTRA-OFFERATOR PARALLELISM**

```sql
SELECT A.id, B.value
FROM A, B
WHERE A.id = B.id
AND A.value < 99
AND B.value > 100
```
INTRA-OPERATOR PARALLELISM

```
SELECT A.id, B.value
FROM A, B
WHERE A.id = B.id
AND A.value < 99
AND B.value > 100
```
### INTRA-OPERATOR PARALLELISM

**Example SQL Query:**

```sql
SELECT A.id, B.value
FROM A, B
WHERE A.id = B.id
AND A.value < 99
AND B.value > 100
```
INTER-OPERATOR PARALLELISM

Approach #2: Inter-Operator (Vertical)
→ Operations are overlapped in order to pipeline data from one stage to the next without materialization.

Also called pipelined parallelism.
**INTER-OPERATOR PARALLELISM**

```
SELECT A.id, B.value
FROM A, B
WHERE A.id = B.id
AND A.value < 99
AND B.value > 100
```

\[\pi \sigma \sigma A \times B\]

for \(r_1 \in \text{outer:}\)

for \(r_2 \in \text{inner:}\)

`emit(r_1 \times r_2)`
**INTER-OPERATOR PARALLELISM**

SELECT A.id, B.value
FROM A, B
WHERE A.id = B.id
AND A.value < 99
AND B.value > 100

for $r_1 \in$ outer:
for $r_2 \in$ inner:
emit($r_1 \natural r_2$)

for $r \in$ incoming:
emit($\pi r$)
**INTER-OPERATOR PARALLELISM**

```sql
SELECT A.id, B.value
FROM A, B
WHERE A.id = B.id
AND A.value < 99
AND B.value > 100
```

```sql
for r ∈ incoming:
    emit(πr)
```

```sql
for r₁ ∈ outer:
    for r₂ ∈ inner:
        emit(r₁⊙r₂)
```
INTER-OPERATOR PARALLELISM

AFAIK, this approach is not widely used in traditional relational DBMSs. → Not all operators can emit output until they have seen all of the tuples from their children.

This is more common in stream processing systems.
OBSERVATION

Using additional processes/threads to execute queries in parallel won't help if the disk is always the main bottleneck.
→ Can actually make things worse if each worker is reading different segments of disk.
I/O PARALLELISM

Split the DBMS installation across multiple storage devices.
→ Multiple Disks per Database
→ One Database per Disk
→ One Relation per Disk
→ Split Relation across Multiple Disks
MULTI-DISK PARALLELISM

Configure OS/hardware to store the DBMS's files across multiple storage devices.
→ Storage Appliances
→ RAID Configuration

This is transparent to the DBMS.
MULTI-DISK PARALLELISM

Configure OS/hardware to store the DBMS's files across multiple storage devices.
→ Storage Appliances
→ RAID Configuration

This is transparent to the DBMS.

RAID 1 (Mirroring)
Some DBMSs allow you to specify the disk location of each individual database.

→ The buffer pool manager maps a page to a disk location.

This is also easy to do at the filesystem level if the DBMS stores each database in a separate directory.

→ The log file might be shared though.
PARTITIONING

Split single logical table into disjoint physical segments that are stored/managed separately.

Ideally partitioning is transparent to the application.
→ The application accesses logical tables and doesn’t care how things are stored.
→ Not always true.
VERTICAL PARTITIONING

Store a table’s attributes in a separate location (e.g., file, disk volume).
Have to store tuple information to reconstruct the original record.

CREATE TABLE foo ( attr1 INT, attr2 INT, attr3 INT, attr4 TEXT );
VERTICAL PARTITIONING

Store a table’s attributes in a separate location (e.g., file, disk volume).
Have to store tuple information to reconstruct the original record.

CREATE TABLE foo (  
    attr1 INT,  
    attr2 INT,  
    attr3 INT,  
    attr4 TEXT  
);
Horizontal Partitioning

Divide the tuples of a table up into disjoint segments based on some partitioning key.
→ Hash Partitioning
→ Range Partitioning
→ Predicate Partitioning

```
CREATE TABLE foo (  
  attr1 INT,  
  attr2 INT,  
  attr3 INT,  
  attr4 TEXT  
);  
```
HORIZONTAL PARTITIONING

Divide the tuples of a table up into disjoint segments based on some partitioning key.
→ Hash Partitioning
→ Range Partitioning
→ Predicate Partitioning

```
CREATE TABLE foo (  
  attr1 INT,  
  attr2 INT,  
  attr3 INT,  
  attr4 TEXT  
);
```
CONCLUSION

Parallel execution is important.
(Almost) every DBMS support this.

This is really hard to get right.
→ Coordination Overhead
→ Scheduling
→ Concurrency Issues
→ Resource Contention
How to embed application logic inside of a DBMS to make things go faster:

→ Stored Procedures
→ User-defined Functions
→ User-defined Types
→ Triggers
→ Views
EXTRA CREDIT

Each student can earn extra credit if they write a encyclopedia article about a DBMS.
→ Can be academic/commercial, active/historical.

Each article will use a standard taxonomy.
→ For each feature category, you select pre-defined options for your DBMS.
→ You will then need to provide a summary paragraph with citations for that category.
EXTRA CREDIT

Each student can earn extra credit if they write an encyclopedia article about a DBMS.

→ Can be academic/commercial, active/historical.

→ Each article will use a standard taxonomy.

→ For each feature category, you select pre-defined options.

→ You will then need to provide a summary paragraph with...
Each student can earn extra credit if they write an encyclopedia article about a DBMS. → Can be academic/commercial, active/historical. Each article will use a standard taxonomy. → For each feature category, you select predefined options for your DBMS. → You will then need to provide a summary paragraph with citations for that category.
Each student can earn extra credit if they write an encyclopedia article about a DBMS. This can be academic/commercial, active/historical. Each article will use a standard taxonomy. For each feature category, you select predefined options for your DBMS. You will then need to provide a summary paragraph with citations for that category.
DBDB.IO

All the articles will be hosted on our new website.
→ I will post the user/pass on Piazza.

I will post a sign-up sheet for you to pick what DBMS you want to write about.
→ If you choose a widely known DBMS, then the article will need to be comprehensive.
→ If you choose an obscure DBMS, then you will have do the best you can to find information.
This article must be your own writing with your own images. You may not copy text/images directly from papers or other sources that you find on the web.

Plagiarism will not be tolerated. See CMU's Policy on Academic Integrity for additional information.